

Appendix for GarmentTracking: Category-Level Garment Pose Tracking

1. VR-Folding Dataset

1.1. Task Definition

In this section, we will describe the details of our task definition in our VR-Folding dataset.

1.1.1 Flattening

Fig. 1 gives some examples of *Flattening* task. The goal of this task is making a garment in randomly crumpled state into a canonical flattened T-pose (see Fig. 1) by a series of actions (*e.g.* grasp, fling). Inspired by GarmentNets [3] and FlingBot [5], we firstly grasp the garment on one randomly selected point to simplify its initial configuration and increase its visibility, then fling the garment with both hands to flatten it efficiently. Specifically, this task can be divided into the following steps:

1. **Grasp with a single hand:** The volunteer will grasp one randomly selected point on the garment with a single hand, and lift it in the air (see sub-figure [1] for Shirt in Fig. 1). This initial configuration is similar to the definition in GarmentNets [3].
2. **Grasp with both hands:** The volunteer will try to grasp two separate points on the garment with two hands simultaneously, and get ready for the following *fling* action (see sub-figure [2] for Shirt in Fig. 1).
3. **Fling:** The volunteer will fling the garment with both hands to erase the wrinkles (see sub-figure [3-7] for Shirt in Fig. 1).
4. Repeat Step 2 and Step 3 until the garment is in flattened T-pose (see sub-figure [7-12] for Shirt in Fig. 1).

1.1.2 Folding

Fig. 2 gives some examples of *Folding* task. This task can be achieved by a series of bimanual pick-and-place actions. We make some general rules for the whole folding process. In order to enhance data variety, we encourage volunteers to fold garments in different ways (*e.g.* fold from the left part or fold from the right part) without violating these rules. Here is the detailed rules of *Folding* for each category:

1. **Shirt:** The volunteer performs 2 (short-sleeve) or 3 (long-sleeve) pick-and-place actions with both hands for *Shirt* (see examples for *Shirt* in Fig. 2). For long-sleeved shirts, the first two actions will make the two sleeves folded, and the last action will make the trunk part folded. For short-sleeved shirts, the first action will fold in half along the left and right direction, and the last action will fold in half along the up and down direction.
2. **Pants:** The volunteer performs 2 pick-and-place actions with both hands for *Pants* (see examples for *Pants* in Fig. 2). The first action will fold in half along the left and right direction, and the last action will fold in half along the up and down direction.
3. **Top:** The volunteer performs 2 pick-and-place actions with both hands for *Top* (see examples for *Top* in Fig. 2). The first action will fold in half along the left and right direction, and the last action will fold in half along the up and down direction.
4. **Skirt:** The volunteer performs 2 pick-and-place actions with both hands for *Skirt* (see examples for *Skirt* in Fig. 2). The first action will fold in half along the left and right direction, and the last action will fold in half along the up and down direction.

1.2. Data Statistics

		Shirt	Pants	Top	Skirt
Folding	# of instances	989	1538	903	461
	# of videos	993	1551	889	463
	# of frames	66830	77479	37214	23427
Flattening	# of instances	1037	1631	1040	467
	# of videos	2145	1720	1063	943
	# of frames	243460	136411	132580	73357

Table 1. Statistics of VR-Folding dataset.

The detailed data statistics of VR-Folding dataset is shown in Tab. 1. In the re-rendering process, we capture frames in Unity at the speed of 10 FPS. Each video only contains one instance, but one instance could occur in multiple videos. Each frame contains 4-view RGB-D images

along with mask and complete garment mesh with NOCS labels. The multi-view RGB-D frames will be filtered with masks and transformed into point cloud. We split the whole dataset into *train*, *val*, *test* subset on ratio [0.8, 0.1, 0.1] by instances, which means that all the instances in videos of *val* or *test* subset are **unseen** during training.

2. GarmentTracking Network

2.1. 3D Feature Extractor

The 3D feature extractor (sparse ResUNet3D) used in our GarmentTracking network is based on FCGF [4]. The sparse 3D convolution operation is implemented by MinkowskiEngine [1]. In our design, the output feature channels of each 3D convolution layer and transpose 3D convolution layer are [64, 64, 128, 256] and [64, 64, 64, 128] respectively. The feature dimension of output per-point feature is 64. Besides, we remove the L2-normalized operation for the final feature, because the relation attention module described below will perform L2-normalization for the feature later. Other hyper-parameters of the network structure are the same as FCGF [4].

2.2. Relation Attention Module (RAM)

The self-attention module and cross-attention module are both based on the same Transformer-like structure called *Relation Attention Module (RAM)* proposed by PTTR [6]. RAM can be formulated as $\text{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, which firstly projects input feature vectors of "Query (Q)", "Key (K)" and "Value (V)" into a latent feature space and then estimates the attention matrix between "Query" and "Key". The attention matrix is then applied to the "Value" feature to obtain the final attention product. Specifically, the self-attention and cross-attention operation can be formulated as Eq. (1), Eq. (2) and Eq. (3):

$$\bar{\mathbf{X}}_1 = \text{Att}(\mathbf{X}_1, \mathbf{X}_1, \mathbf{X}_1) \quad (1)$$

$$\bar{\mathbf{X}}_2 = \text{Att}(\mathbf{X}_2, \mathbf{X}_2, \mathbf{X}_2) \quad (2)$$

$$\hat{\mathbf{X}} = \text{Att}(\bar{\mathbf{X}}_2, \bar{\mathbf{X}}_1, \bar{\mathbf{X}}_1) \quad (3)$$

where \mathbf{X}_1 and \mathbf{X}_2 stands for features of previous frame and current frame respectively. $\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2$ are the self-attention features and $\hat{\mathbf{X}}$ is the cross-attention fusion feature. It is worth noting that RAM is a lightweight transformer, which only brings small computing overhead for the whole pipeline. The layer number and the head number of RAM are both 1. The feature dimension in the middle layer of RAM is 64. The MLP channels in the final feature layer is [64, 128, 128]. The dimension of the final fusion per-point feature is 128.

2.3. NOCS Refiner

The detailed network structure of NOCS Refiner is shown in Tab. 2. We use ReLU as activation function and

use batch normalization in all the MLPs. During inference, the *PC Refiner* will refine the raw NOCS class logits for each frame. However, we find that the *Mesh Refiner* only needs very few steps to achieve reasonable results during tracking. So we only enable the *Mesh Refiner* for the first few frames during long-term tracking for inference, which could increase the stability of the final outputs. For *Folding* task, we enable it only in the first frame. For *Flattening* task, we enable it in the first 15 frames in the video.

Name	Feature Dimensions of MLP
PC-PointNet	[326, 256, 256, 1024]
Mesh-PointNet	[3, 64, 128, 1024]
Mesh Fusion MLP	[2112, 512, 512, 1024]
Mesh Refine MLP	[1024, 512, 256, 6]
PC Refine MLP	[2304, 1024, 512, 192]

Table 2. The detailed network structure of NOCS Refiner.

2.4. Details of Training and Inference

The batch size in training is 16 for all of our experiments. During training, we randomly select two continuous frames in the same video as input. During inference, we use the predicted per-point NOCS coordinate and the refined canonical mesh of the previous frame as the input of the current frame. We remove the static frames (*i.e.* frames without garment movement) and only track the moving frames (*i.e.* frames that contain garment movement) in videos during inference.

3. Additional Experiment Results

3.1. Noise Parameters for Robustness Experiment

In the robustness experiment (Sec. 5.3.3) of the main paper, we augment the point-cloud NOCS coordinates of the first frame with a global scaling factor s_{pc} , a global offset \mathbf{o}_{pc} , and Gaussian noise standard deviation δ . Besides, we also augment the canonical mesh with global scaling factor s_{mesh} . The detailed setting of these noise parameters is shown in Tab. 3.

3.2. Qualitative Results on VR-Folding Dataset

Fig. 3 shows additional qualitative results on **unseen** instances in VR-Folding dataset. We use GarmentNets [3] as the baseline here. **Please watch the video in the supplementary files for more elaborate examples.**

3.3. Real-world Experiments

We collect some real-world RGB-D (640×480 , 30FPS) videos of garment manipulation with 4 Realsense L515 [2] LiDAR cameras. Before recording, We perform calibration

Noise Level	s_{pc}	\mathbf{o}_{pc}	δ	s_{mesh}
1x	$[0.8, 1.2]^3$	$[0, 0.1]^3$	0.05	$[0.8, 1.2]^3$
2x	$[0.6, 1.4]^3$	$[0, 0.2]^3$	0.10	$[0.6, 1.4]^3$
3x	$[0.4, 1.6]^3$	$[0, 0.3]^3$	0.15	$[0.4, 1.6]^3$

Table 3. The noise parameters in the robustness experiment. $[a, b]^3$ indicates a 3-D vector (*i.e.* x, y, z axis) in which each dimension is uniformly sampled from $[a, b]$.

(*i.e.* intrinsic and extrinsic parameters) for the four L515 cameras. During recording, we ask the volunteer to repeat the same garment manipulation actions in VR-Folding dataset on **novel** garments in real world. After recording, we uniformly drop 3/4 of the frames in the raw videos which makes the final videos all at 7.5FPS. Nextly, we perform post-processing for the recorded data. We firstly generate partial point cloud from the depth map of each camera then merge the multi-view partial point cloud into one single point cloud. Finally, we ask volunteers to segment the merged point cloud and only keep the garment part in the point cloud.

In order to narrow the gap between simulation and real world, we re-train our model and GarmentNets with pure depth information (*i.e.* no RGB in point cloud) and perform zero-center operation for the input point cloud during training. During inference, we directly use the GarmentNets prediction (*i.e.* canonical coordinates and mesh) as the first-frame pose.

Fig. 4 and Fig. 5 show some qualitative results on **unseen** garments in our collected real-world data. Our method can directly track pose for novel garments in the real-world with a model trained only on our simulated data, and exhibit more stability compared to the baseline (*i.e.* GarmentNets). **Please watch the video in the supplementary files for more elaborate examples.**

4. Broader Impact

- Since our dataset only contains the tasks associated with the daily garments, our work will not facilitate injury to living beings directly.
- We paid great attention to protect our volunteers’ privacy in the process of collecting data.
- We respect human rights in all parts of our work.
- Our work only focuses on two tasks about deformable garments, so it’s impossible to use our work to develop or extend harmful forms of surveillance.
- Our work causes no damage to the environment, because the dataset we built and used is collected with a VR system.

- The garments appear in our VR-Folding dataset are all generated by software, so our dataset is unlikely to deceive people in real life.

References

- [1] Nvidia Minkowski Engine. <https://github.com/NVIDIA/MinkowskiEngine>. 2
- [2] RealSense L515 LiDAR camera. <https://www.intelrealsense.com/lidar-camera-l515/>. 2
- [3] Cheng Chi and Shuran Song. Garmentnets: Category-level pose estimation for garments via canonical space shape completion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3324–3333, 2021. 1, 2
- [4] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8958–8966, 2019. 2
- [5] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022. 1
- [6] Changqing Zhou, Zhipeng Luo, Yueru Luo, Tianrui Liu, Liang Pan, Zhongang Cai, Haiyu Zhao, and Shijian Lu. Pptr: Relational 3d point cloud object tracking with transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8531–8540, 2022. 2

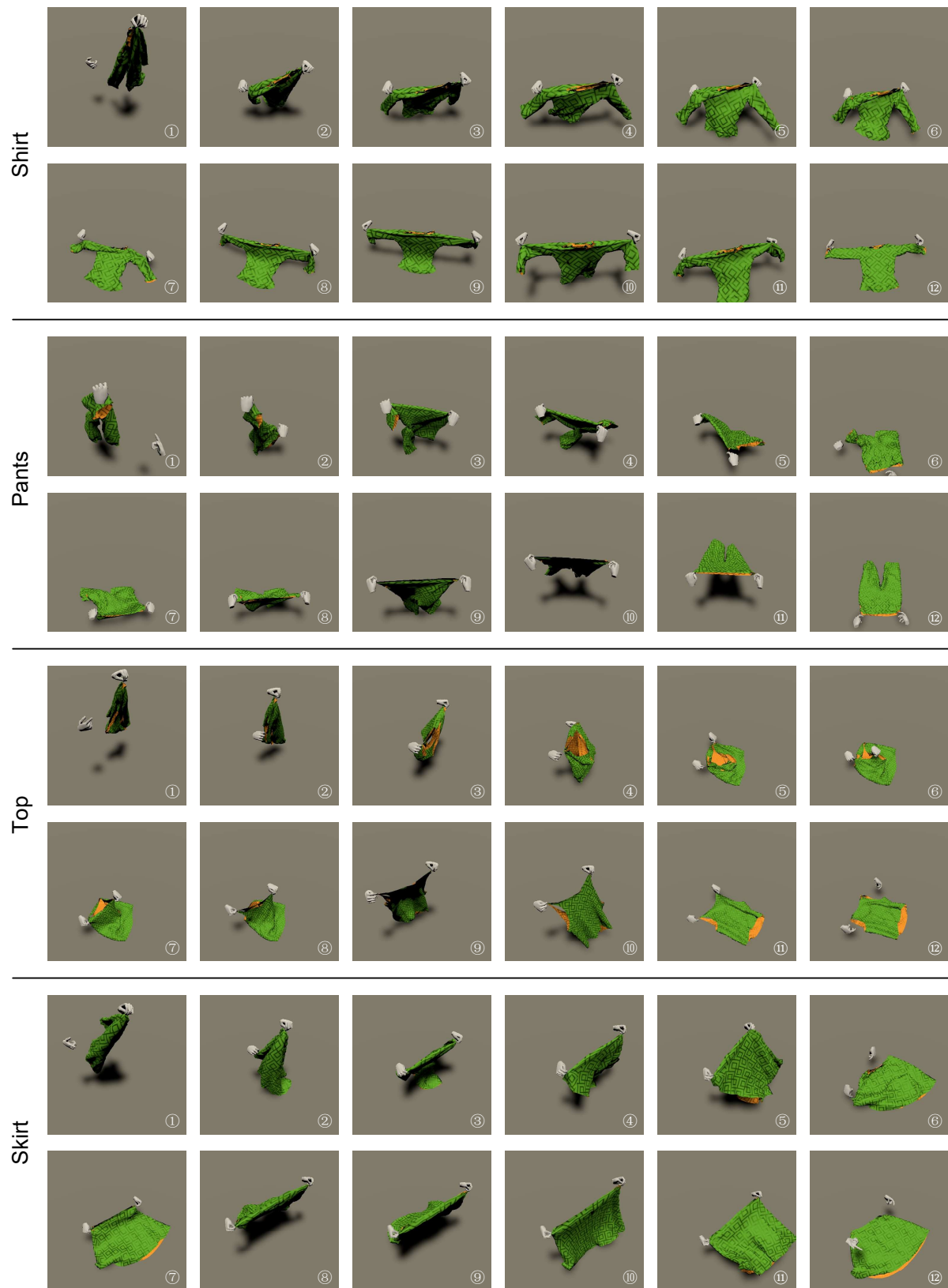


Figure 1. The examples of **Flattening** task.

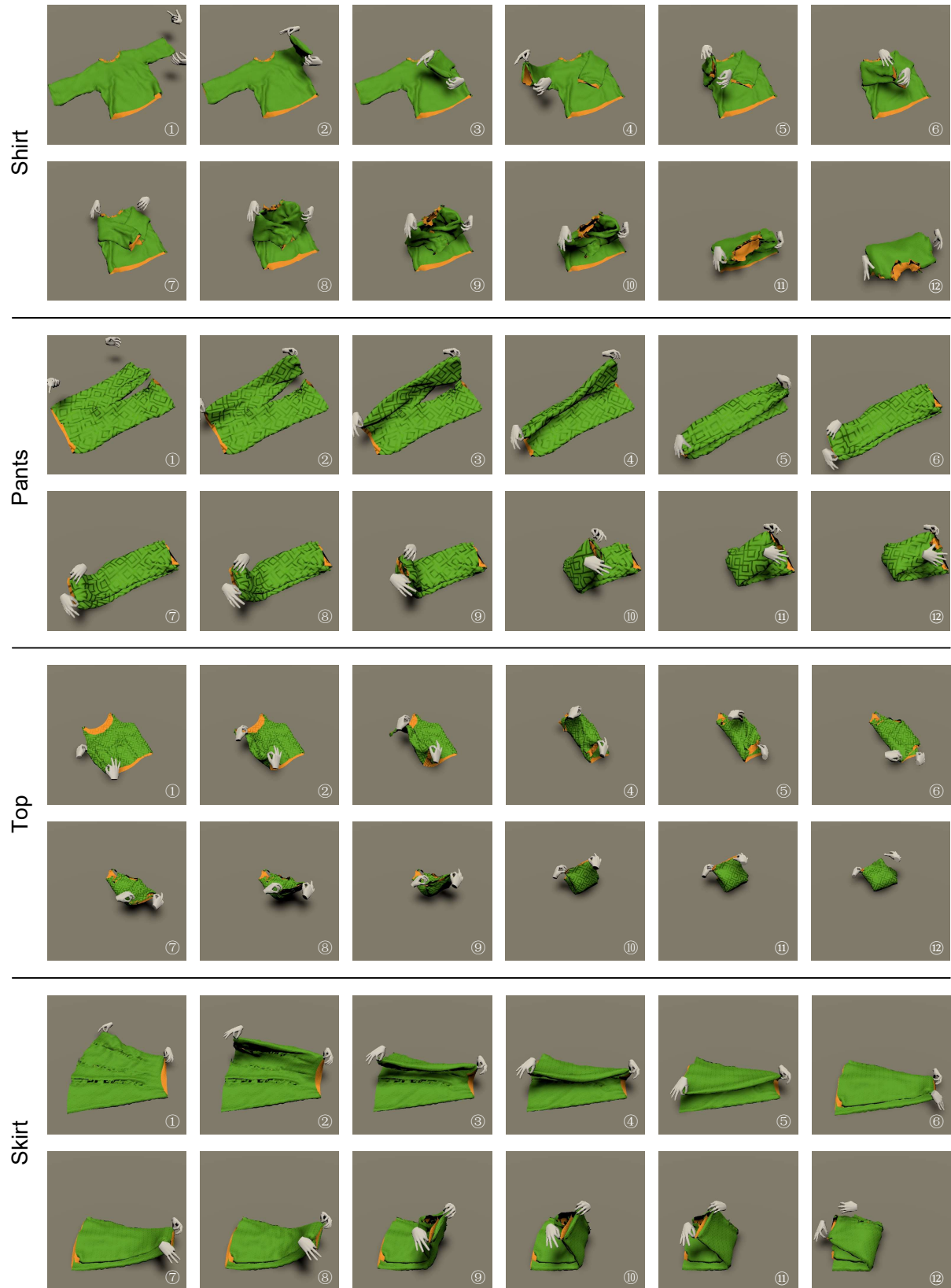


Figure 2. The examples of **Folding** task.

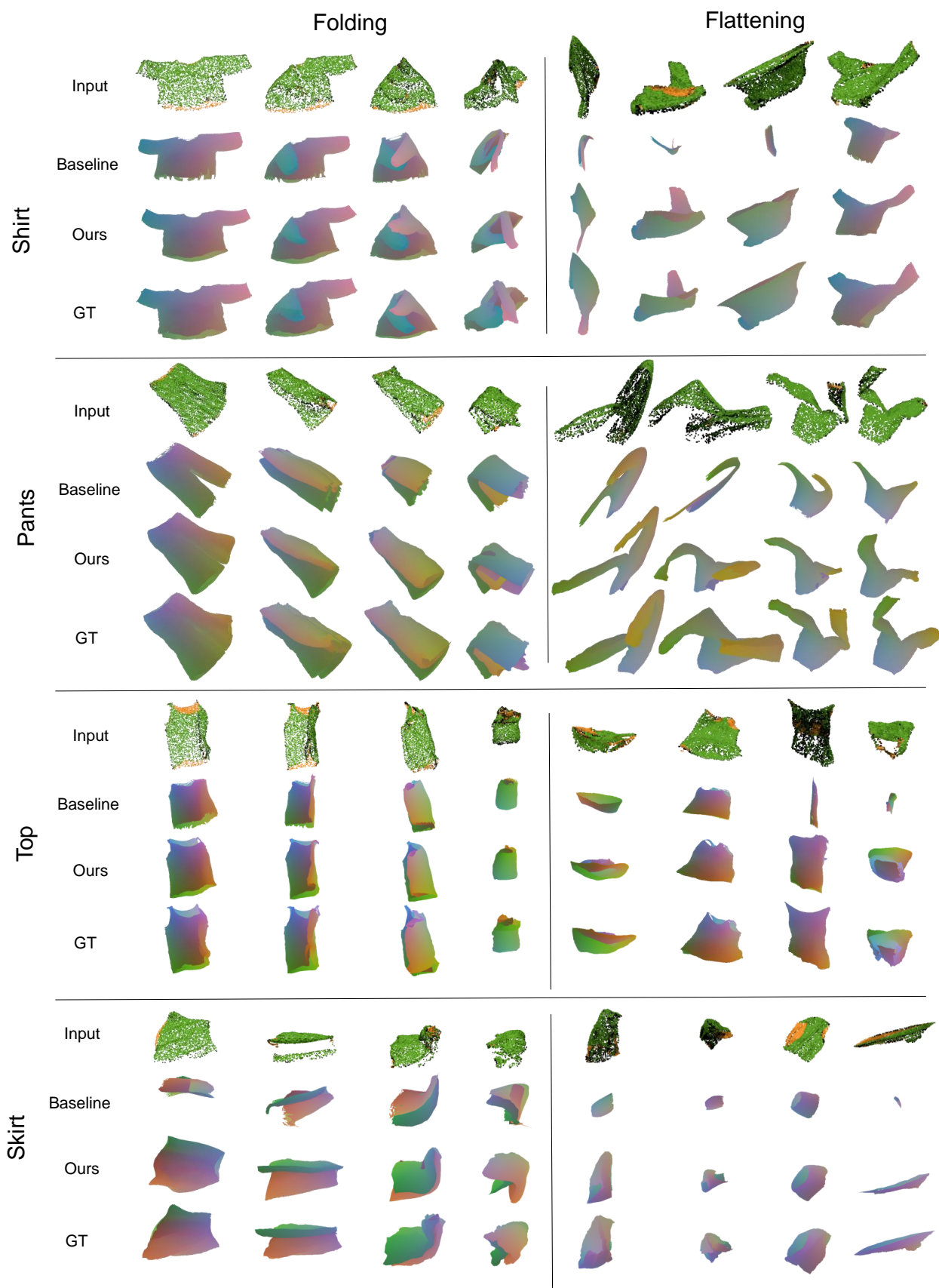


Figure 3. The additional qualitative results on **unseen** instances in VR-Folding dataset. **Please watch the video in the supplementary files for more elaborate examples.**

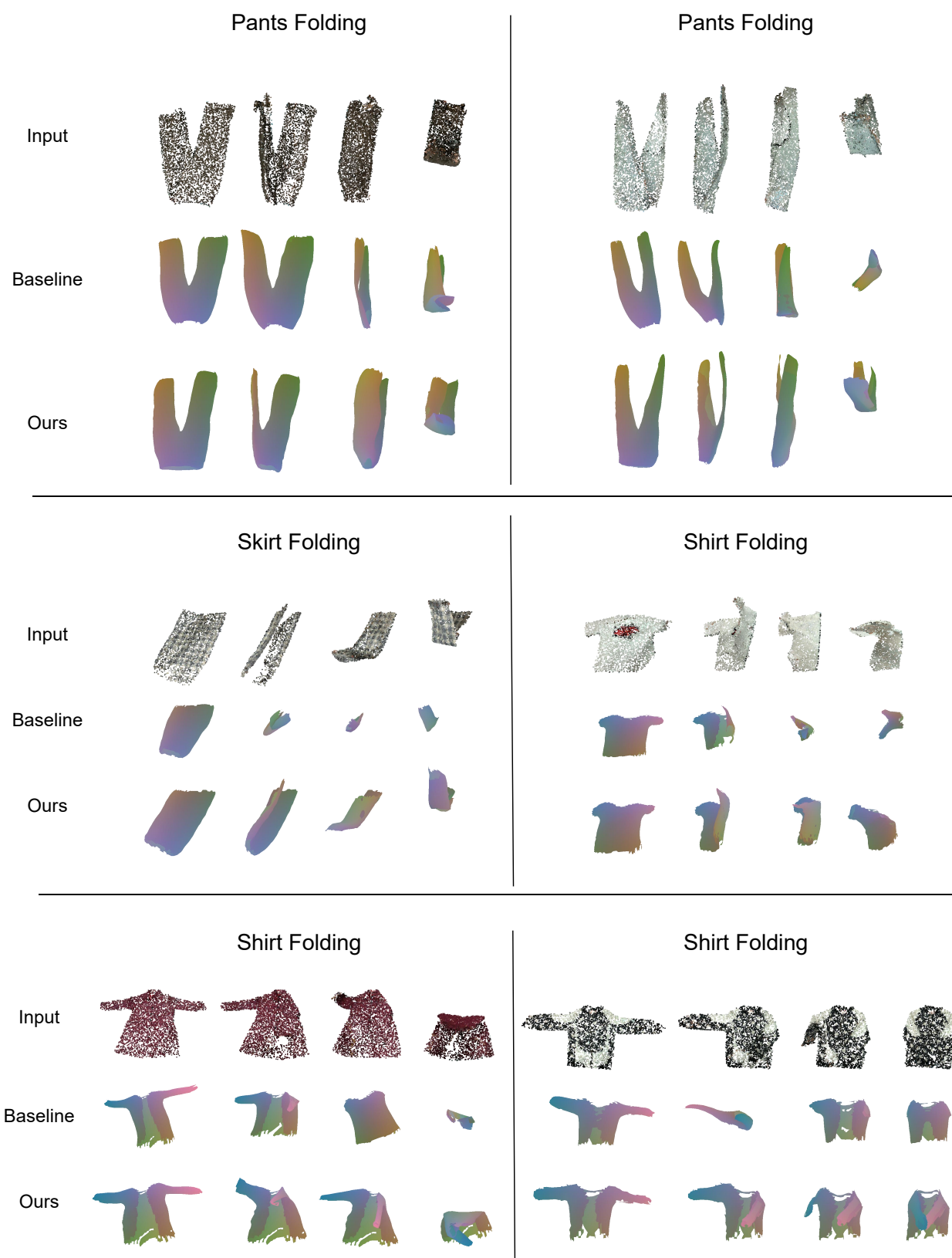


Figure 4. The qualitative results on **unseen** instances for *Folding* task in **real-world** data. **Please watch the video in the supplementary files for more elaborate examples.**

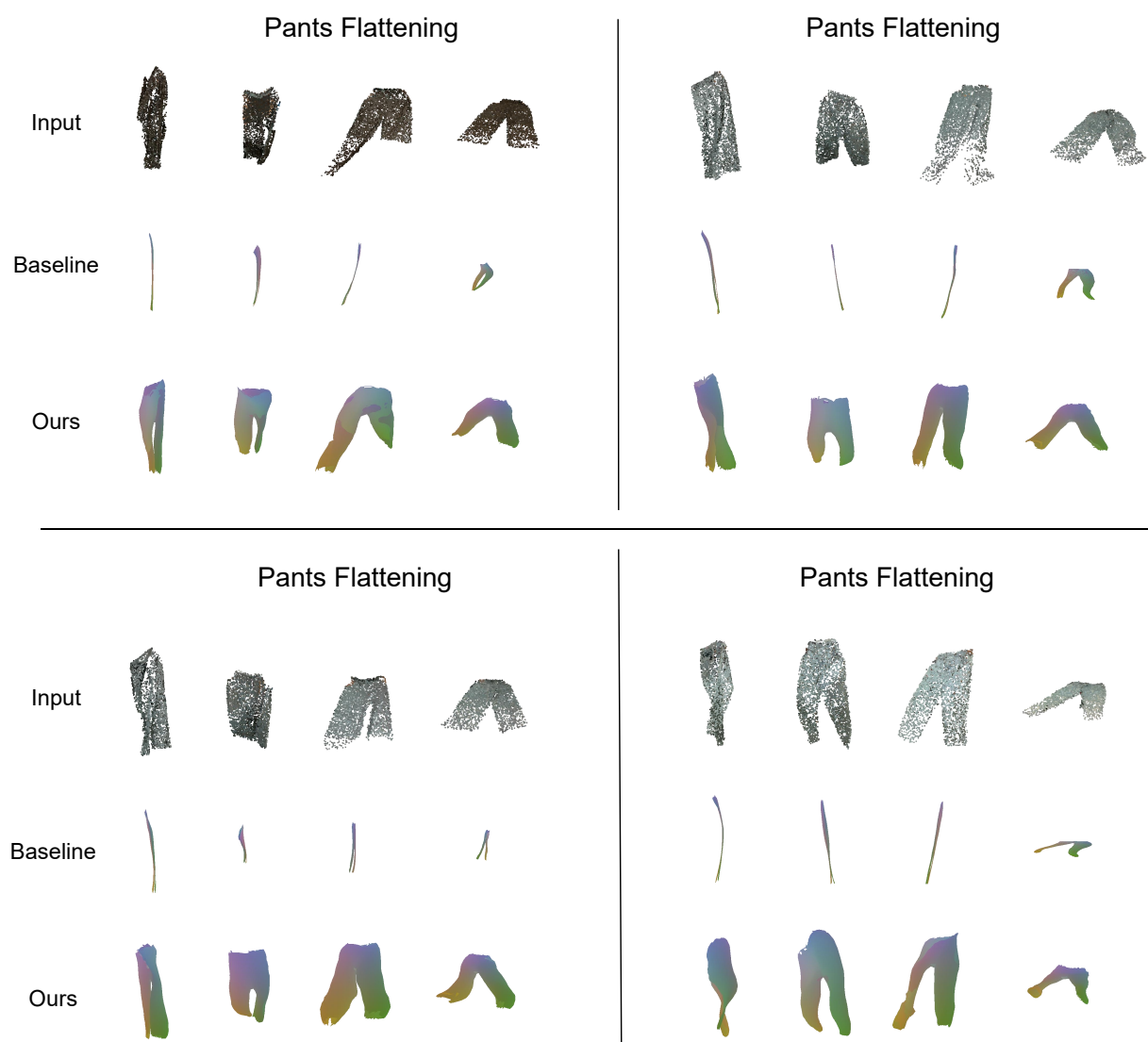


Figure 5. The qualitative results on **unseen** instances for *Flattening* task in **real-world** data. **Please watch the video in the supplementary files for more elaborate examples.**